Face Reconstruction with RGB-D images

Final Report (Group 21)

Schubert, Daniel Vinzenz Konstantin Chan, Tong Yan Luo, Chang

February 10, 2022

1 Motivation and Idea

The goal of our project is to fully reconstruct a human face, including the shape and expression, and skin color, from an RGB-D image. Typical multiple-view reconstruction via bundle adjustment often relies on a number of frames for a well-reconstructed object, especially when the resolution of the camera is low. To address this, we explore another method for high-quality face reconstruction and aim to reconstruct faces through optimizing the parameters of a parametric face model. t Here is a summary of the major steps in our pipeline and its flowchart in Fig.1.

1. RGB-D data collection and pre-processing
2. Detect facial landmarks in RGB image, and backproject them to world space
3. Extract the 3D coordinates of the corresponding landmarks from Basel Face Model
4. Estimate the pose of face model with the 3D landmarks by Procrustes Analysis
5. Optimize the pose and parameters of face shape and expression to the landmarks in a sparse term
6. Further optimize the parameters and face color on all vertices in a dense term

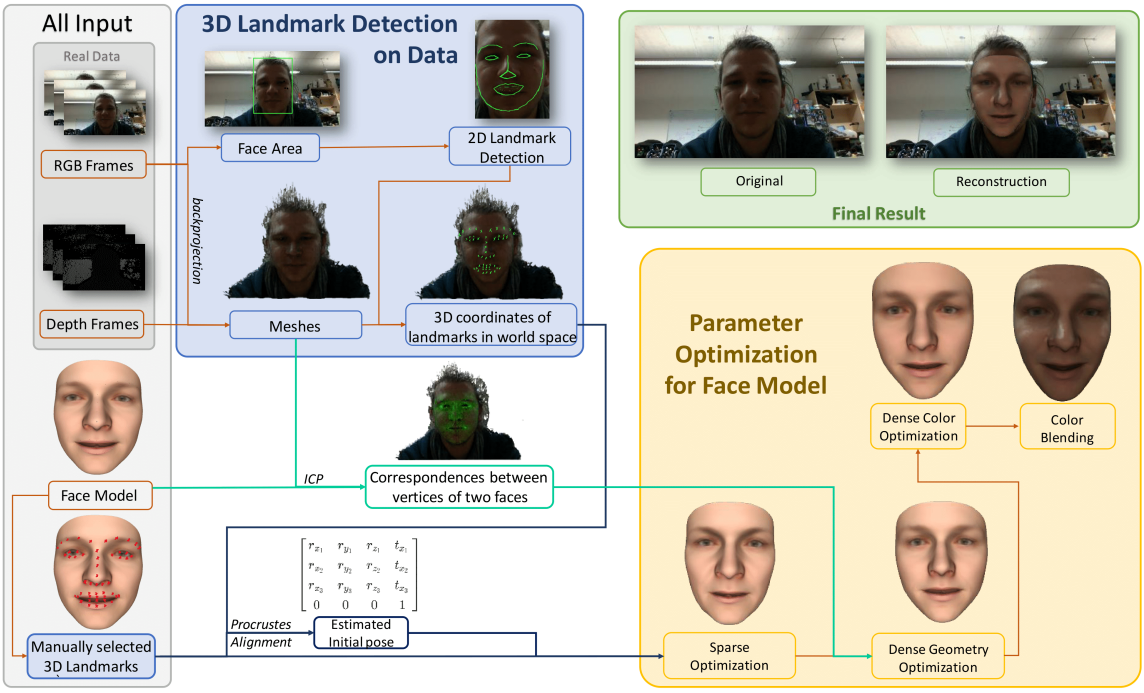


Figure 1: Overview of pipeline

2 Related Works

Reconstruction of facial geometry from images has been approached multiple times. Solutions differ by the input they use, such as RGB-D data[8] or only RGB input[9] and by the target of their work.

2.1 Dlib for Facial Landmark Detection

To locate faces in data, face detection algorithm is needed. Dlib is an open-source library containing machine learning algorithms for C++ [4]. We can make use of the library to detect face position and then further estimate the landmarks on the face.

2.2 Datasets

Even though there are lots of research on face reconstruction, most research does not release their datasets publicly. And some of them reconstruct faces merely based on RGB images but no depth information, which is out of the scope of our project. As a result, there are very limited publicly-available face datasets. We have tried out all the datasets we found, but neither one of them suits our needs. The only RGB-D dataset we found was FaceGrabber [5]. It is an application that captures faces in RGB-D streams from Kinect. In the early stage, we have worked on this dataset; however, we later found that they did not provide the transformation between color images and depth images. Thus, we could not simply project the color on a reconstructed point cloud. More importantly, we were not able to locate the detected facial landmarks in the world space. In the end, we decided to capture our own dataset with RGB-D cameras. Details on data collection will be explained in the next section.

2.3 Basel Face Model (BFM)

We use the Basel Face Model [10] 2019 as the basis for our generated representation of the face. It is a 3D Morphable Model (3DMM), which is a principle component based model that provides principle components for face shape, face expression and vertex color, which we would like to optimize. It provides 199 principal components for both shape and color, and 100 components for expression. The weight of these principal components are the target in our optimization task.

2.4 PCL for finding correspondences

In order to perform dense term optimization (which will be introduced in more detail in the following sections), we need to get the correspondences between the point on Basel Face Model and the point on the mesh we constructed from color map and depth map. PCL library [6] is used to carry out this task. Moreover, PCL also provides the Iterative Closest Point (ICP) implementation to register two point clouds, which can be exploited to get a preciser pose matrix.

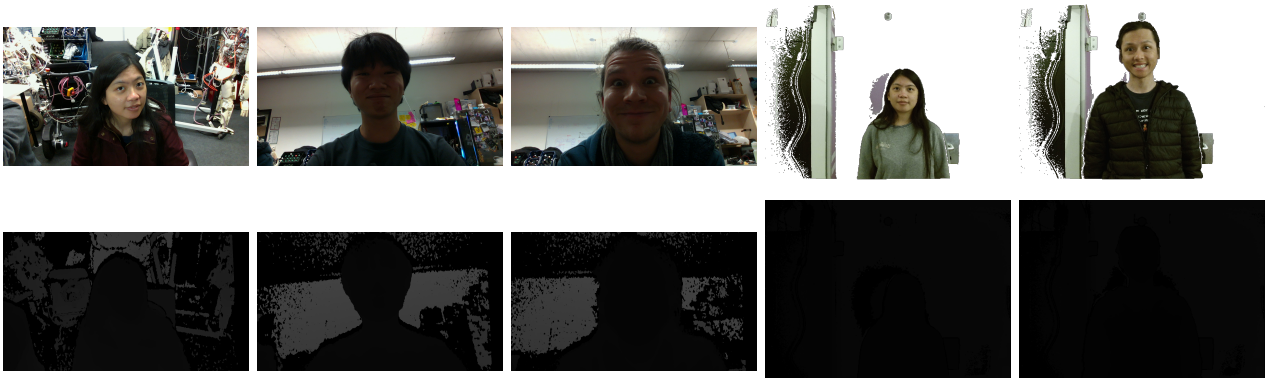
3 Method

The goal of our project is to reconstruct a face via optimizing a parametric face model. To achieve this, we need to align the face model to our RGB-D data in world space first before optimizing the shape, expression and face color parameters of the model. Here, we can make use of facial landmarks for alignment. In this section, we will explain our pipeline in detail.

3.1 Data Collection and Preprocessing

A Microsoft Kinect v2 and an Intel RealSense L515 camera were available in our group, so we captured data from both cameras. During data collection and preprocessing, we used libraries, such as Kinect SDK, that are available for each camera to match the RGB image to the corresponding depth image.

Hence, no further alignment is needed in our software. For Kinect v2 data, the background color is also removed roughly during data collection for better presentation because there are some artifacts; yet, this step is not necessary. The resulting .png files of RGB images and 16-bit greyscale depth data from both cameras were used as input for our algorithm. Fig.2 shows some sample data after preprocessing.



(a) (b) (c) (d) (e)

Figure 2: Same sample data we captured. (a)-(c) are captured by RealSense. (d)-(e) are captured by Kinect v2.

Besides, due to the high resolution of the image captured by RealSense, a direct landmark detection will be time-consuming, hence we additionally employs OpenCV [3] Deep Neural Network (DNN) Face Detection to get the face area, and the cropped face area will be further used in facial landmarks detection.

3.2 Facial Landmarks

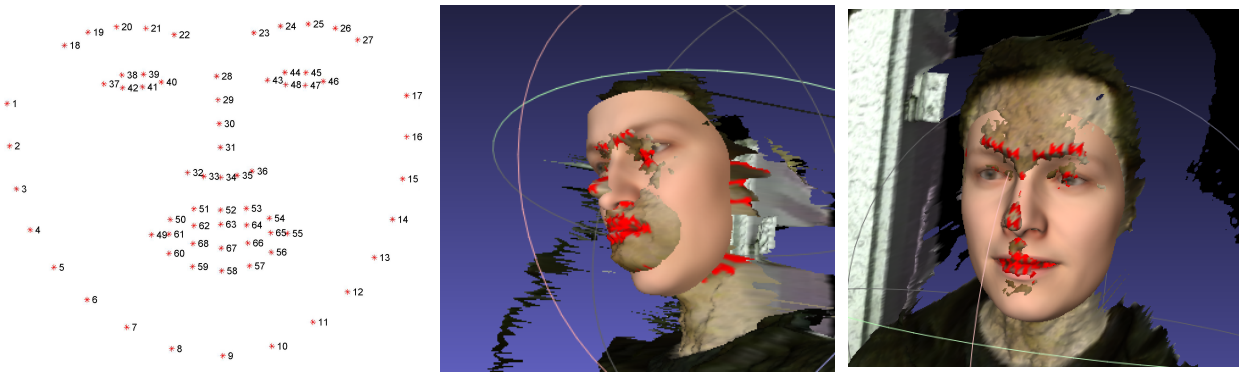
In order to speed up the reconstruction process, we align the face model and optimize the parameters of the face model according to facial landmarks first. These landmarks get detect in our input data and corresponding landmarks are manually select among the vertices of the face model. We use the iBug/300-w 68 landmark annotations [7] for our purpose.

3.2.1 Landmark Detection in our data

We make use of Dlib [4] to detect the 68 facial landmarks in our RGB images. However, we found that in the 3d reconstruction, the outline landmarks of a face are often marked on the neck instead of around the chin. This causes incorrect alignment in the later stage of our pipeline. Thus, we decided to remove the 17 landmarks located on the outline of the face shape, leaving the 51 landmarks on the face. Fig.3 compares the coarse alignment with and without the 17 face outline landmarks. The 51 remaining landmarks are then backprojected into 3D space by using the depth data and the known intrinsic camera parameters.

3.2.2 Landmark Extraction for Basel Face Model

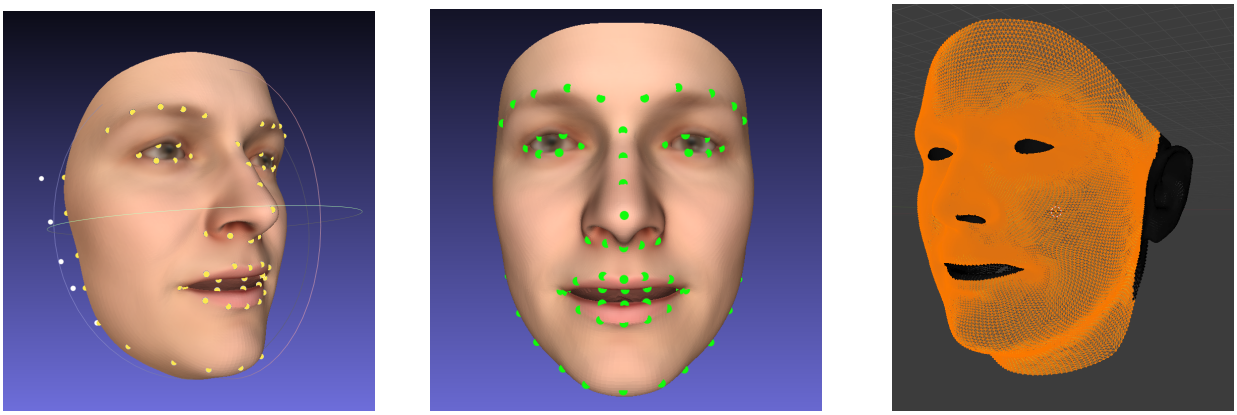
We use the Basel Face Model as the parametric model for the reconstruction of the face in later stages. To align the landmarks of the model to our RGB-D-data, we have to manually select the corresponding landmarks in the model first. We found a dataset online that gave us the vertex indices for the old 2009 version of the BFM [2]. However, the vertex list did not correspond to the latest 2019 version of the BFM that we would like to use. To resolve this, we generated the 3D coordinates of landmarks of the mean face on the 2009 version and used this to find the closest vertices in the new version of Basel Face Model. We generated a list of Vertex IDs for both the normal Basel Face Model (Face including



(a) Landmarks given by Dlib. (b) With all 68 landmarks. (c) Without face outline.

Figure 3: Alignment with and without the landmarks of face outline.

ears) and the face-only version, which we preferably used for our project. During our development, we noticed that the quality of the landmark annotations drastically impacted the quality of our fitting results. We subsequently decided to manually redefine the position of all landmarks on the lips, the nose and the inner part of the eyebrows (see Fig. 4b) because they do not totally match the landmarks estimated by Dlib. This results in much improved fitting of the model to the real-world data.



(a) Landmarks on Basel Face Model 2019. (White - 2009, Yellow 2019) (b) The final and manually adjusted positioning of the landmarks on Basel Face Model. (c) Only the vertices highlighted in orange participate in the dense optimization process.

Figure 4: Landmarks on the Basel Face Model

3.3 Coarse Alignment with Procrustes Analysis

After the 3D coordinates of the landmarks for both our data and the mean face of Basel Face Model are collected, we want to get an initial pose of the face model in world space so that we can speed up the optimization step later. Here, we align the Basel Face Model (as source) to our data (as target) using Procrustes Analysis. Since the face model is not in the same scale as our data, we also need to compute the proper scale first before we estimate the pose. The scale is given by the following formula:

$$scale = \sqrt{\frac{\sum_{i=1}^{n} ||t_i - \sigma_t||^2}{\sum_{i=1}^{n} ||s_i - \sigma_s||^2}}$$

where t_i and s_i are the vertices on the target point cloud and the source point cloud respectively, and σ_t and σ_s are the corresponding means.
The scale factor is then applied to the source vertices before computing the rotation through SVD

decomposition. Translation is computed by the difference between the means of two point clouds. After Procrustes Analysis, we are able to find the estimated scale, rotation and translation of the face model.

3.4 Parameter Optimization for Face Model

After the coarse alignment step, we perform optimizations on the model parameters to fit of the model to the scanned input. The target of optimization is acquiring three parametric parameters (ω_{shape}, $\omega_{expression}$, ω_{color}), which have respectively 199, 100 and 199 principal components.

3.4.1 Sparse Term

Our sparse optimization aligns the landmark vertices of the face model s_k to the landmark vertices of our data input t_k.

$$E_{sparse} = \sum_{k}^{51} ||s_k - t_k||_2^2$$

For this, we create a non-linear least squares problem and use Ceres solver[1] to optimize the shape and expression parameters of the Basel Face Model. We divide the cost by the scale of the model to make sure that it is scale invariant. We also multiply the regularization weights of the parameters with the number of landmarks to allow for a flexible amount of landmarks.

3.4.2 Dense Term

To further optimize the parameters of the face model, we subsequently perform a dense optimization. For this we use PCL to match every point of the face model with its current parametrization to the closest point in the scanned input point cloud. We optimize the shape and expression parameters to minimize the distance between every selected vertex and the corresponding point of the scanned data. To avoid malformed shapes, we selected a subset of vertices which participate in the dense optimization. As can be seen in fig.4c, we specifically omitted the eyes, nostrils, the inner mouth and also the inner part of the lips from this process.
Since the BFM has quite a dense mesh, we found that it is sufficient to only use a fraction of all vertices.
Therefore, to speed up the cost minimization process, we only create the cost residual blocks for a randomly chosen fraction of 4% of the vertices within our subset. Our dense cost function therefore is:

$$E_{dense} = \sum_{k \in V} ||s_k - t_k||_2^2$$

where s_k is the vertex on the model, t_k the corresponding point of the 3D scan and V the subset of vertices to perform the optimization on. We consider extending the dense cost approach to a point-to-plane method for future improvements.
We noticed that modifying the parameters of the BFM changes the shape enough to actually change which point of the scanned input a vertex should be mapped to. To benefit from this, we decided to create an extra loop that repeats this whole dense optimization step multiple times with re-matched point correspondences.

3.4.3 Color Term

While the previous optimizations only considered the geometry of the face model, we decided to include the color in our reconstruction method. Since the BFM also provides principal components for the color of the vertices, we decided to replicate the dense optimization term to the color space. We use Ceres to minimize the color difference between each vertex and its corresponding raw data point.

$$E_{color} = \sum_{k \in V} ||C_{sk} - C_{tk}||_2^2$$

We noticed that while the face model gives good generalization and allows for matching the base appearance of a face, we wanted our method to specifically mirror the individual subject we scanned. For this, where available, we mix in the raw color of the corresponding input point additionally to the vertex color coming from the Basel face model. The resulting color is used for the representations shown in the report.

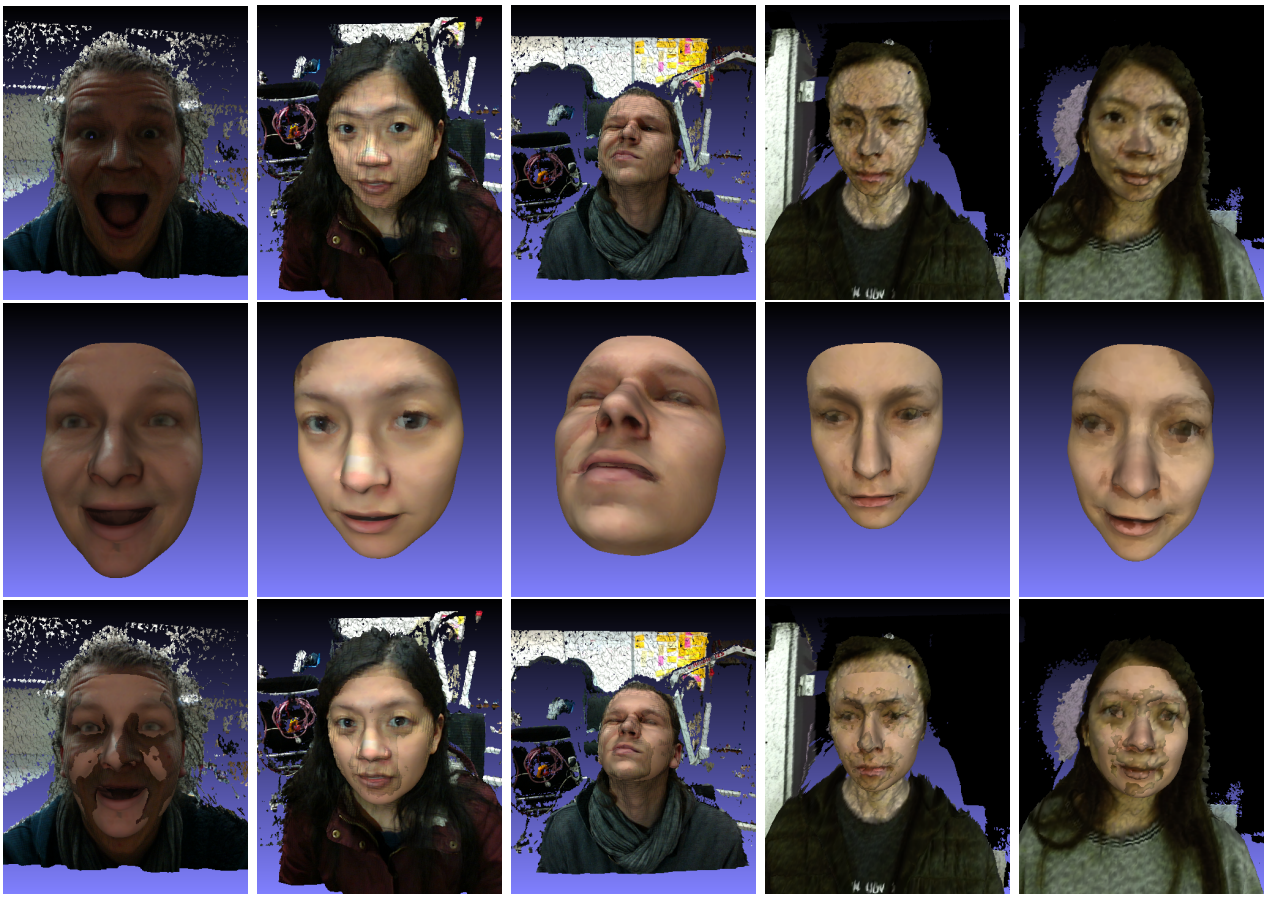
3.4.4 Regularization

For both of the optimization terms mentioned above, we add individually weighted regularization, which makes sure the individual parameters don't go overboard and allows us to tweak them individually, as following describes, W is a constant weighting factor, and ω is refering to ω_{shape}, $\omega_{expression}$ or ω_{color}.

$$E_{regularizer} = Constraints \times W \times \omega$$

4 Results

To generate our final reconstruction, we only need to feed the preprocessed RGB-D frames into our program. Fig.5 shows a few examples of the reconstructed face via our pipeline.



(a) (b) (c) (d) (e)

Figure 5: Final reconstruction of different people, face expression and face angles. Row-1: the backprojected meshes; row-2: the reconstructed face; row 3: combined view of row-1 and row-2 in MeshLab. (a)-(c) are reconstructed from RealSense data. (d)-(e) is reconstructed from Kinect v2 data.

5 Analysis

For identifying the performance from each sub-step, we made distance plots Fig. 6, where the distance is the distance from that point on the BFM model to its corresponding in the input data, with points that are close to each other marked in blue and distant points marked in red.

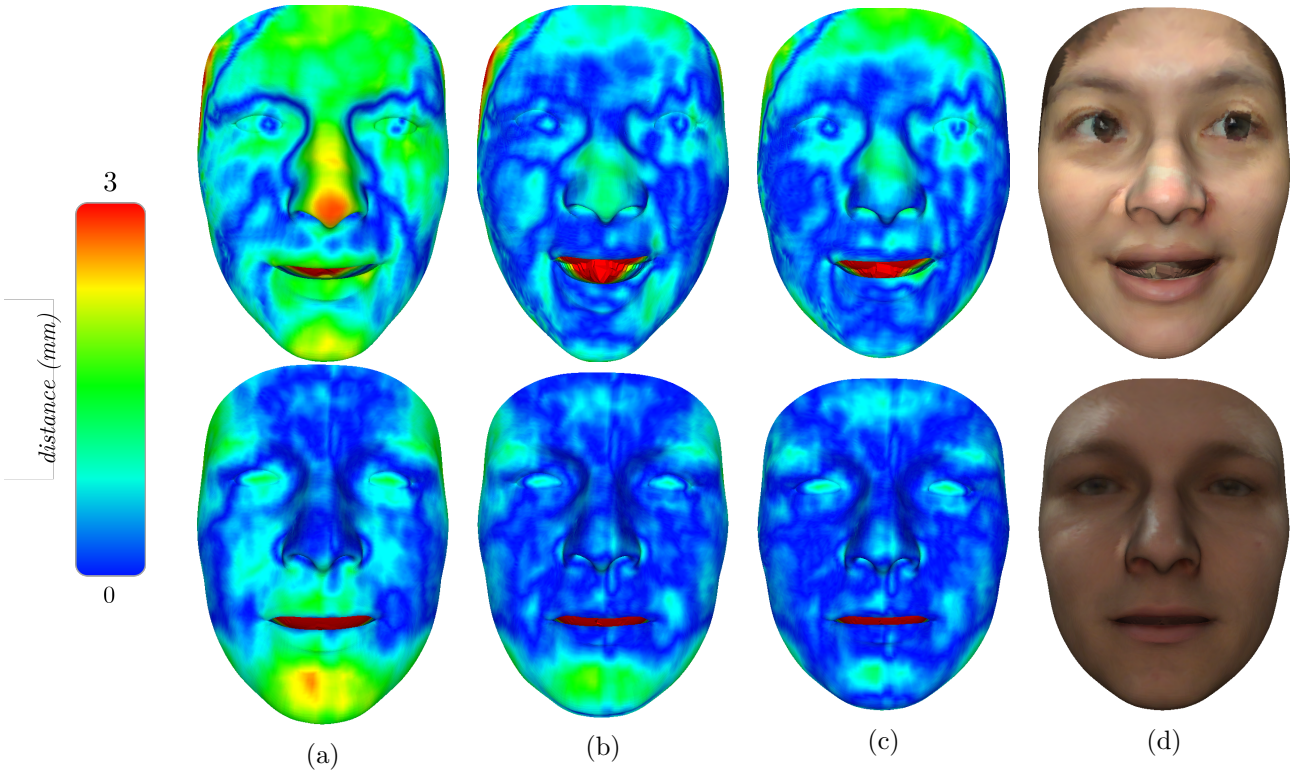


Figure 6: Distance maps between the reconstruction and the real face. (a) Distance map after Procrustes Analysis. (b) Distance map after Sparse Term optimization. (c) Distance map after Dense Term optimization. (d) Reconstructed face.

We can observe that at the stage when the Procrustes analysis is just completed, the BFM is basically aligned with the input data, but there are still some areas in the chin, tip of the nose and forehead where the model does not match the input. After sparse term optimization we can see these parts were well improved, and for dense term optimization, we get a better fit on chin and cheek but degrade around mouth region. The forehead, in this case, reacts a bit problematic to the hair. In most cases, we saw a significant improvement in the fit after applying the dense optimization.

Additionally, we can find the whole alignment and optimization does present a better performance on the European male than on the Asian female dataset by comparing the first row and second row of Fig. 6. And we also want to illuminate that the performance difference between the high res-data and low-res data, by observing the output distance map of Kinect v2 feeding (see Fig.7), it's easy to notice that there are some freaks on the face model, the reason for that the data has less vertices than BFM Model, therefore for a region of BFM vertices they posses a common public corresponding point.

6 Conclusion

We show that our method is a viable approach for reconstructing facial geometry from RGB-D camera data by optimizing parametric face models such as the Basel Face Model after we have made several adjustments on better landmark selection. However, there is also a few improvements we can consider. To further improve our approach, we suggest the following considerations:

6.1 Facial Landmarks

The DLib model used in our approach does not perform well under some circumstances. Fig.8 shows one example of an extreme facial expression with a widely-opened mouth. The landmarks of the lower lip in this case were projected inside of the mouth leading to bad performance of the optimizations. This problem can be resolved by using a more modern facial feature detection model.

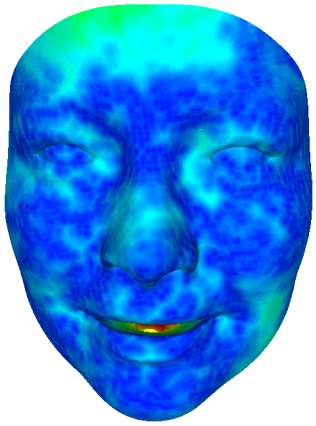


Figure 7: Distance map of Low-Res feeding.

Figure 8: Incorrect facial landmarks provided by Dlib.

6.2 Performance improvements towards a real-time solution

We believe that it would be possible to further develop our approach into a real-time solution for facial tracking. For this, we would propose to use the GPU implementation of the facial landmark detection first, which could reduce the time drastically. Secondly, the optimized parameters of the facial model should be passed to the next frame as a pre-initialization of the model. As the identity of the tracked subject can be assumed to stay constant, only the expression parameters would need continuous optimization while the face shape parameter could be optimized over multiple frames.

Runtime Analysis	
Sub-step	Consumed time
OpenCV & BFMContainer & VirtualSensor Initialization	0.143s
OpenCV DNN face detection & Dlib landmark detection	0.109s
Procrustes Analysis & ICP & Correspondences	1.263s
SparseTerm Optimization	0.209s
DenseTerm Geometry Optimization (3 iterations)	1.834s
DenseTerm Color Optimization	0.298s
Output	3.982s
Process one frame without output 3.856s	

Table 1: Runtime analysis on Windows 11, Ryzen 7 5800X, GTX1160Ti, 32GB

References

[1] Sameer Agarwal, Keir Mierle, and Others. Ceres solver. http://ceres-solver.org.

[2] Anil Bas. Landmarks for basel face model (3dmm).

[3] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] Davis King. Dlib C++ face landmark detection example.

[5] D. Merget, T. Eckl, M. Schwrer, P. Tiefenbacher, and G. Rigoll. Capturing facial videos with kinect 2.0: A multithreaded open source tool and database. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016.

[6] Radu Bogdan Rusu and Steve Cousins. 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[7] Christos Sagonas. ibug - resources - Facial point annotations, 2016.

[8] Justus Thies, Michael Zollhfer, Matthias Niener, Levi Valgaerts, Marc Stamminger, and Christian Theobalt. Real-time expression transfer for facial reenactment. *ACM Transactions on Graphics*, 34(6):1–14, November 2015.

[9] Justus Thies, Michael Zollhfer, Marc Stamminger, Christian Theobalt, and Matthias Niener. Face2Face: Real-time Face Capture and Reenactment of RGB Videos. *arXiv:2007.14808 [cs]*, July 2020. arXiv: 2007.14808.

[10] Clemens Blumer Bernhard Egger Marcel Lthi Sandro Schnborn Thomas Gerig, Andreas Morel-Forster and Thomas Vetter. Morphable face models - an open framework, 2018.